aws-orgs

Mar 23, 2020

Contents:

CHAPTER 1

Getting started with aws-orgs

A configuration management tool set for AWS Organizations.

Full documentation is available at https://aws-orgs.readthedocs.io/en/latest

1.1 Features

- Ensure state of AWS Organizations and IAM resourses per yaml formatted specification files.
- Configure AWS Organizations resources:
  - organizational units
  - service control policies
  - account creation and organizational unit placement
- Centrally manage IAM access across AWS Organization accounts:
  - IAM users/groups in a central *Auth* account
  - customer managed IAM policies
  - IAM roles and trust delegation in organization accounts

1.2 Installation

Python virtual environment (recommended):

```
source ~/path_to_my_venv/bin/activate
pip install aws-orgs
```

Editable copy in venv:

```
git clone https://github.com/ucopacme/aws-orgs
pip install -e aws-orgs/
```

Uninstall:

```
pip uninstall aws-orgs
```

1.3 Configuration quick start

Run the `awsorgs-spec-init` script to generate an initial set of spec-files:

```
awsorgs-spec-init
```

This generates an initial `config.yaml` spec files under `~/.awsorgs`. Edit these as needed to suit your environment.

See `--help` option for full usage.

1.4 Console Scripts

`aws-orgs` provides the following python executibles:

awsorgs Manage recources in an AWS Organization.

awsaccounts Manage accounts in an AWS Organization.

awsauth Manage users, group, and roles for cross account access in an AWS Organization.

awsloginprofile Manage AWS IAM user login profile.

All commands execute in `dry-run` mode by default. Include the `--exec` flag to affect change to AWS resources. Run each of these with the '–help' option for usage documentation.

```
awsorgs report
awsorgs organization
awsorgs organization --exec

awsaccounts report
awsaccounts create [--exec]
awsaccounts alias [--exec]

awsaccounts invite --account-id ID [--exec]
# from invited account:
awsorgs-accessrole --master_id ID [--exec]

awsauth report
awsauth report --users
awsauth report --delegations
awsauth report --credentials --full
awsauth report --account ucpath-prod --users --full

awsauth users [--exec]
awsauth delegations [--exec]
awsauth local-users [--exec]
```

(continues on next page)

(continued from previous page)

```
awsloginprofile maryanne
awsloginprofile maryanne --new
awsloginprofile maryanne --reset
awsloginprofile maryanne --disable-expired --opt-ttl 48
```

Author Ashley Gould (agould@ucop.edu)

Version 0.3.0

CHAPTER 2

Configuration Basics

There are two aspects os `aws-orgs` configuration:

1. per-user cli configuration - *config.yaml*
2. site wide resource specifcation - *Spec files*

The `awsorgs-spec-init` tool is provided to *Bootstrap your initial configuration*.

2.1 config.yaml

Most CLI commands make use of a per-user config file for basic paramaters. The default location is `~/.awsorgs/config.yaml`. This file supplies the values for required cli option parameters:

```
--spec-dir
--master-account-id
--auth-account-id
--org-access-role
```

Example:

```
# AWSORGS Configuration Parameters

# Path to yaml spec files directory.  Any yaml files under this dirctory
# (recursive) are parsed as spec files.
spec_dir: ~/.awsorgs/spec.d

# An AWS role name which permits cross account access to all accounts.
org_access_role: awsauth/OrgAdmin

# Use an alternate role from master account when setting up a new account.
#org_access_role: OrganizationAccountAccessRole

# AWS account Id for the Organization master account.  This must be in quotes.
```

(continues on next page)

(continued from previous page)

```
master_account_id: '121212121212'

# AWS account Id for the Central Auth account.  This must be in quotes.
# This Central Auth account can be the same as the Master account.
auth_account_id: '343434343434'
```

Copy this file to your home directory (or run `awsorgs-spec-init`) and edit parameter values to suit your AWS Organization.

2.2 Spec files

AWS-ORGS makes use of a complex of YAML formatted resource specification files. The spec files are used by aws-orgs commands to deploy and maintain AWS accounts and IAM resources across your Organization.

A set of example spec files gets installed on your system when you run `awsorg-spec-init`. Each or the example spec files contains complete documentation of spec attributes. Edit these to suit your AWS Organization.

2.2.1 Example Spec Files

shared

common.yaml

```
---
# AWSORGS configuration parameters common to all operations.

# AWS account Id for the Organization master account.  This must be in␣
↪quotes.
master_account_id: '121212121212'

# AWS account Id for the Central Auth account.  This must be in quotes.
auth_account_id: '343434343434'

# Email domain to use for account creation if the accounts['Email'] field
# is not explicitly specified.
default_domain: example.com

# Default Organizational Unit.  Any accounts in the Organization not
# explicitly assigned to an Organizational Unit are placed here.
default_ou: root

# Default Organization Service Control Policy.  This is managed by AWS and
# should not be modified or deleted.  This is attached to all Organizational
# Units.
default_sc_policy: FullAWSAccess

# This string is prepended to all IAM resource 'path' attributes.
default_path: awsauth

# Default SMTP email server to use when sending email messages.
default_smtp_server: smtp.example.com
```

(continues on next page)

(continued from previous page)

```
# The organization administration team.  These contacts are references when
# generating email messages.
org_admin_team: administration
```

teams.yaml

```
---
# Teams Specification.
#
# Teams provides a labeling system for tracking folks within your␣
↪organization
# who are responsible for managed accounts and users.
#
# Each team spec has the following attributes:
#   Name (str):                    The name of the user
#   Description (str):             Team description
#   BusinessContacts (list(str)):  Email addresses of business contacts
#   TechnicalContacts (list(str)): Email addresses of technical contacts

teams:
- Name: administration
  Description: AWS Organization Administration
  BusinessContacts:
  - ashley@example.com
  TechnicalContacts:
  - admins@example.com
- Name: development
  Description: Application developers
  BusinessContacts:
  - kalila@example.com
  TechnicalContacts:
  - dimna@example.com
```

awsorgs

organizational_units.yaml

```
---
# Organizational Unit Specification.
#
# This specification maps the Organization's structure and assigns policies␣
↪and
# accounts to organizational units.
#
# Each organizational_unit spec (OU) has the following attributes:
#   Name (str):     The name of the OU (required)
```

(continues on next page)

(continued from previous page)

```
#   Ensure (str):   One of 'present' (default) or 'absent'.  Setting to
#                   'absent' will cause the OU to be deleted but
#                   only if no accounts are still assigned to the OU.
#   Accounts (list(str)):
#                   List of account names assigned to this OU.
#   SC_Policies (list(str)):
#                   List of Service Control Policies attached to this OU.
#   Child_OU (list(organizational_unit)):
#                   List of child Organizational Units (recursive structure).

organizational_units:
# the root OU must be defined
- Name: root
  Accounts:
  - master-account
  - central-auth
  Policies:
  Child_OU:
  - Name: applications
    Accounts:
    - dev1
    SC_Policies:
    - LimitAWSRegions
```

service_control_policies.yaml

```
---
# Service Control Policy Specification
#
# Defines custom Service Control Policies which can then be attached
# to Organizational Units.
#
# Each service control policy spec (SCP) has the following attributes:
#   Name (str):         The name of the SCP.
#   Ensure (str):       One of 'present' (default) or 'absent'. Setting
#                       to 'absent' will cause the SCP to be deleted, but
#                       only if it not attached to any Organizational Unit.
#   Description (str):  The policy description.
#   Statement (list(dict)):
#                       List of IAM policy statements applied to the SCP.

sc_policies:
- PolicyName: LimitAWSRegions
  Ensure: present
  Description: Limit the AWS regions where users can deploy resources
  Statement:
  - Sid: DenyAllRegionsOutsideUS
    Effect: Deny
    NotAction:
    - iam:*
    - organizations:*
    - route53:*
    - budgets:*
```

(continues on next page)

(continued from previous page)

```
    - waf:*
    - cloudfront:*
    - globalaccelerator:*
    - importexport:*
    - support:*
    Resource: '*'
    Condition:
      StringNotEquals:
        aws:RequestedRegion:
        - us-east-1
        - us-east-2
        - us-west-1
        - us-west-2
```

awsaccounts

accounts.yaml

```
---
# Accounts Specification.
#
# List of managed AWS accounts in the Organization.
#
# Each account spec the following attributes, all of type 'str':
#   Name:   The name of the account
#   Email:  The email address used when creating a new account.  This
#           address must be unique in all AWS.  If omitted, we combine
#           the account name and the default_email_domain.
#   Team:   A label for tracking folks within your organization associated
#           with a this account. This must match a 'Name' attribute from a
#           team spec.
#   Alias:  String to use for the account alias.  Defaults to 'Name' in
#           lower case.

accounts:
  - Name: master-account
    Team: administration
    Alias: master
    Email: master-account@example.com
  - Name: central-auth
    Team: administration
    Alias: auth
    Email: central-auth@example.com
  - Name: dev1
    Team: development
```

awsauth

users.yaml

```
---
# Users Specification
#
# List of IAM users managed within the Central Auth account.
#
# Each user has the following attributes all of type 'str':
#   Name:   The name of the user - required.
#   Ensure: One of 'present' (default) or 'absent'.
#           Setting to 'absent' will cause the user to be deleted.
#   Email:  The email address with which the user can be contacted.
#   Team:   A label for associating the IAM user with a project team.
#   Path:   Set a path element to the IAM user resource.

users:
- Name: ashley
  Email: ashley@example.com
  Team: administration
- Name: kalila
  Email: kalila@example.com
  Team: development
- Name: dimna
  Email: dimna@example.com
  Team: development
```

groups.yaml

```
---
# Group Specification
#
# List of IAM group resources managed within the Central Auth account.
#
# Each group has the following attributes:
#  Name (str):      The group name.
#  Ensure (str):    One of 'present' (default) or 'absent'. Setting to
#                   'absent' will cause the group to be deleted, but
#                   only if the group contains no users.
#  Members (list(str), 'ALL'):
#                   List of IAM users who are members of this group.
#                   If set to 'ALL', all managed users in the Central
#                   Auth account become members.
#  ExcludeMembers (list(str)):
#                   If 'Members' attribute is set to 'ALL', any users
#                   listed in 'ExludeMembers' are excluded from the group.
#  Policies (list(str)):
#                   List of IAM policies to attach to this group.

groups:
- Name: all-users
  Members: ALL
  Policies:
  - UserSelfService
```

(continues on next page)

(continued from previous page)

```
- Name: admins
  Members:
  - ashley
- Name: devops
  Members:
  - kalila
  - dimna
```

delegations.yaml

```
---
# AWS Auth Delegations Specification
#
# A delegation is a complex of IAM resources which combine to allow
# users in the ('trusted') Auth account to access and manipulate
# resources in one or several or the other ('trusting') accounts in
# the Organization.  This is accomplished by managing a delegation
# role in the trusting accounts which contains a trust policy naming
# the Auth account as 'principal', and by assigning 'assume role'
# policies to a managed group in the Auth account for each trusting
# account within the scope of the delegation specification.
#
# Each delegation spec has the following attributes:
#   RoleName (str):     The name of the IAM role created in trusting␣
↪accounts.
#   Ensure ('present'[default]|'absent'):
#                       Determines whether the IAM role exists or not.
#                       Setting to 'absent' deletes delegation roles in
#                       trusting accounts and removes assume role policies
#                       from the trusted group.
#   Description (str):  A decription applied to the IAM role.
#   TrustingAccount (list(str), 'ALL'):
#                       List of trusting accounts within the scope of the
#                       delegation.  If set to 'ALL', all accounts in the
#                       Organization are include in the delegation.
#   ExcludeAccounts (list(str)):
#                       If 'TrustingAccount' attribute is set to 'ALL',
#                       any accounts listed in 'ExludeAccounts' are
#                       excluded from the delegation.
#   TrustedGroup (str): The IAM group in the Auth account in which to assign
#                       assume role policies for this delegation.
#   TrustedAccount (str):
#                       The account Id to use as principle in service roles.
#   RequireMFA (bool):  When set to 'True' (the default), add
#                       a condition to the trust policy requiring users
#                       assuming the delegation role to have valid MFA token.
#   Duration (int):     MaxSessionDuration time in seconds. Default is 3600.
#   Policies (list(str)):
#                       List of IAM policies to attach to the delegation role
#                       in the trusting accounts.
#   PolicySet (str):    Name of the policy set to attach to the delegation␣
↪role
```

(continues on next page)

(continued from previous page)

```
#                       Incomplatible with "Policies".

delegations:

- RoleName: AccountAdmin
  Ensure: present
  Description: Full access to all services
  TrustingAccount: ALL
  ExcludeAccounts:
  - master-account
  TrustedGroup: admins
  RequireMFA: True
  Policies:
  - AdministratorAccess

- RoleName: Developer
  Ensure: present
  Description: Allow developers access in dev1 account
  TrustingAccount:
  - dev1
  TrustedGroup: developers
  RequireMFA: True
  PolicySet: Developer
```

policy_sets.yaml

```
# A Policy Set is a list of IAM policies, either AWS managed or customer
# managed, which taken in composit, define the permissions available to
# a particular job function, such as "Developer" of "SecurityAuditor".
#
# Each policy set spec has the following attributes:
#   Name (str):             the policy set name
#   Descriptsion (str):     describes the scope of the policy set
#   Tags (list(dict)):      list of tags to apply to delegation roles made␣
↪from
#                           this policy set
#   Policies (list(str)):   list of IAM policy names

policy_sets:
- Name: Developer
  Description: Access for application developers.
  Tags:
  - Key: jobfunctionrole
    Value: True
  Policies:
  - PowerUserAccess
  - IAMReadOnlyAccess
- Name: TesterPolicySet
  Description: Access for testers
  Tags:
  - Key: jobfunctionrole
    Value: True
```

(continues on next page)

(continued from previous page)

```
  Policies:
  - ReadOnlyAccess
  - ReadS3Bucket
```

custom_policies.yaml

```
---
# IAM Custom Policy Specification
#
# List of IAM policy definitions.  Custom policies are created in accounts in
# which the policy is attached to least one group or delegation role.
#
#
# Each custom policy spec (SCP) has the following attributes:
#   Name (str):         The name of the SCP.
#   Description (str):  The policy description.
#   Statement (list(dict)):
#                       List of IAM policy statements applied to the SCP.

custom_policies:
- PolicyName: UserSelfService
  Description: Allow users to manage thier own account and credentials
  Statement:
  - Sid: AllowAllUsersToListAccounts
    Effect: Allow
    Action:
    - iam:ListAccountAliases
    - iam:ListUsers
    - iam:ListUserPolicies
    - iam:ListAttachedUserPolicies
    - iam:GetAccountSummary
    - iam:ListGroups
    - iam:ListGroupPolicies
    - iam:ListAttachedGroupPolicies
    - iam:GetGroup
    - iam:GetGroupPolicy
    - iam:ListMFADevices
    Resource: "*"
  - Sid: AllowIndividualUserToSeeAndManageTheirOwnAccountInformation
    Effect: Allow
    Action:
    - iam:ListGroupsForUser
    - iam:ChangePassword
    - iam:CreateAccessKey
    - iam:CreateLoginProfile
    - iam:DeleteAccessKey
    - iam:DeleteLoginProfile
    - iam:GetAccountPasswordPolicy
    - iam:GetLoginProfile
    - iam:ListAccessKeys
    - iam:UpdateAccessKey
    - iam:UpdateLoginProfile
```

(continues on next page)

(continued from previous page)

```
    - iam:ListSigningCertificates
    - iam:DeleteSigningCertificate
    - iam:UpdateSigningCertificate
    - iam:UploadSigningCertificate
    - iam:ListSSHPublicKeys
    - iam:GetSSHPublicKey
    - iam:DeleteSSHPublicKey
    - iam:UpdateSSHPublicKey
    - iam:UploadSSHPublicKey
    Resource: arn:aws:iam::*:user/*/${aws:username}
  - Sid: AllowIndividualUserToListTheirOwnMFA
    Effect: Allow
    Action:
    - iam:ListVirtualMFADevices
    - iam:ListMFADevices
    Resource:
    - arn:aws:iam::*:mfa/*
    - arn:aws:iam::*:user/*/${aws:username}
  - Sid: AllowIndividualUserToManageTheirOwnMFA
    Effect: Allow
    Action:
    - iam:CreateVirtualMFADevice
    - iam:DeactivateMFADevice
    - iam:DeleteVirtualMFADevice
    - iam:RequestSmsMfaRegistration
    - iam:FinalizeSmsMfaRegistration
    - iam:EnableMFADevice
    - iam:ResyncMFADevice
    Resource:
    - arn:aws:iam::*:mfa/${aws:username}
    - arn:aws:iam::*:user/*/${aws:username}
  - Sid: BlockAnyAccessOtherThanAboveUnlessSignedInWithMFA
    Effect: Deny
    NotAction:
    - iam:*
    Resource: "*"
    Condition:
      BoolIfExists:
        aws:MultiFactorAuthPresent: 'false'
```

local_users.yaml

```
---
# Local User Specification
#
# IAM user resources can be deployed into managed accounts.  Local users
# are typically accociated with a service such as SES or S3.  Such users
# do not have a login profile.
#
# Each local user spce has the following attributes:
#   Name (str):       The name of local IAM user.
#   Ensure ('present'[default]|'absent'):
```

(continues on next page)

(continued from previous page)

```
#                     Ensures whether the local user exists or not.
#   Description (str):
#                     A decription applied to the local IAM user.
#   Team (str):       A label for associating the IAM user with a project␣
↪team.
#                     This must match a 'Name' attribute from a team spec.
#   Path (str):       Path prefix for the IAM user resource name. (optional)
#   Account (list(str), 'ALL'):
#                     List of accounts in which the user is deployed.
#                     If set to 'ALL', the local user will be created in
#                     all accounts in the Organization.
#   Policies (list(str)):
#                     List of IAM policies to attach to the local user.

local_users:
- Name: local-service-user
  Description: Local service user
  Team: development
  Path: service
  Account: All
  Policies:
  - ReadOnlyAccess
```

The default spec directory is `~/.awsorgs/spec.d`. If you choose a non-default location, be sure to update the `spec_dir` parameter in your `config.yaml`.

Note: Keep your spec files under version control!

2.3 Bootstrap your initial configuration

`aws-orgs` provides a helper script `awsorgs-spec-init`. This script generates an initial `config.yaml` and a full set of example spec files. By default these are installed under `~/.awsorgs`:

```
> awsorgs-spec-init
find ~/.awsorgs
~/.awsorgs/config.yaml
~/.awsorgs/spec.d/accounts.yaml
~/.awsorgs/spec.d/common.yaml
~/.awsorgs/spec.d/custom_policies.yaml
~/.awsorgs/spec.d/delegations.yaml
~/.awsorgs/spec.d/groups.yaml
~/.awsorgs/spec.d/local_users.yaml
~/.awsorgs/spec.d/orgs.yaml
~/.awsorgs/spec.d/policy-sets.yaml
~/.awsorgs/spec.d/service_control_policies.yaml
~/.awsorgs/spec.d/teams.yaml
~/.awsorgs/spec.d/users.yaml
```

Run `awsorgs-spec-init --help` for options on how to install to alternate locations.

CHAPTER 3

Concepts and Workflow

Ahh! So much to tell. So little motivation!

CHAPTER 4

Working with Spec Files

Note: Work in Progress

4.1 Shared Specs

common.yaml Top level spec attributes common to all tools.

teams.yaml these are used as attributes in other spec objects:

- accounts
- users
- local_users

4.2 `awsorgs`

organizational_units.yaml

service_control_policies.yaml

4.3 `awsaccounts`

accounts.yaml

4.4 awsauth

users.yaml

groups.yaml

delegations.yaml

policy_sets.yaml

custom_policies.yaml

local_users.yaml

CHAPTER 5

AWS Organizations

5.1 Provissioning Accounts - `awsaccounts`

Prerequisites:

- admin access to auth account
- spec file setup
  - `~/.awsorgs/config.yaml` for awsorgs configuration parameters
  - spec files in `spec_dir` directory which is defined in `config.yaml`
  - create at least one satelite account (see `awsaccounts`)

5.1.1 AWS account alias

Commands used:

- awsaccounts alias
- awsaccounts alias –exec
- awsaccounts report

spec files impacted:

- ~/.awsorgs/config.yaml
- spec_dir/account.yaml

Actions Summary:

- Define `org_access_role` in `~/.awsorgs/config.yaml`
- Define `spec_dir/account.yaml`
- `awsaccounts alias`

- `awsaccounts alias --exec`
- `awsaccounts report`

Define `org_access_role` in `~/.awsorgs/config.yaml`

Edit `~/.awsorgs/config.yaml`:

```
org_access_role: OrganizationAccountAccessRole
```

Show current account aliases

Run `awsacccounts report`:

```
(py36) [jhsu@scrappy-aws ~]$ awsaccounts report


__________________________
Active Accounts in Org:

Name:                Alias               Id:             Email:
account-abcaws1   acct-abcaws1      123456789011    mail1@yahoo.com
account-abcaws2   acct-abcaws2      123456789022    mail2@yahoo.com
account-abcaws3   acct-abcaws3      123456789033    mail3@yahoo.com
```

Edit `spec_dir/account.yaml`

Change `account-abcaws3` alias to `accnt-abcaws3`:

```
- Name: account-abcaws3
Team: team-abcaws3
Alias: accnt-abcaws3
Email: mail3@yahoo.com
```

Dryrun awsaccounts alias

Run `awsaccount alias`:

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts alias

[dryrun] awsorgs.utils: INFO     resetting account alias for account 'account-abcaws3
↪' to 'accnt-abcaws3'; previous alias was 'acct-abcaws3'
```

Exec awsaccounts alias

Run `awsaccount alias --exec`:

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts alias --exec

awsorgs.utils: INFO     resetting account alias for account 'account-abcaws3' to
↪'accnt-abcaws3'; previous alias was 'acct-abcaws3'
```

awsaccounts report

Run `awsaccount report`:

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts report



_________________________
Active Accounts in Org:

Name:                Alias               Id:             Email:
account-abcaws1   acct-abcaws1      123456789011    mail1@yahoo.com
account-abcaws2   acct-abcaws2      123456789011    mail2@yahoo.com
account-abcaws3   accnt-abcaws3     123456789011    mail3@yahoo.com
```

5.2 Organizational Units and Service Control Policies - `awsorgs`

Note: Under Custruction

CHAPTER 6

Centralized IAM Access

6.1 Users and Groups - `awsauth users`

Prerequisites:

- admin access to auth account
- spec file setup
  - install template spec files
  - spec files under git
  - site specific paramaters defined in common.yaml
  - configure ~.awsorgs/config.yaml
  - create at least one satelite account (see awsaccounts)

Commands used:

- git diff
- awsauth users
- awsauth users –exec
- awsauth report –users

Spec files impacted:

- users.yaml
- groups.yaml
- custom_policies.yaml

Actions Summary:

- *Report users and groups in all accounts*
- *Create an IAM user and group, and add the user to the group*

- *Attach a IAM managed policy to your group*
- *Attach a IAM custom policy to your group*
- *Modify attached custom policy*
- *Detach policies, users from group*
- *Delete group, delete users*

6.1.1 Report users and groups in all accounts

Run `awsauth report` command with `--users` flag:

```
$ awsauth report --users
_____________________________________________
IAM Users and Groups in all Org Accounts:
_____________________________
Account:    Managment
Users:
- arn:aws:iam::123456789011:user/awsauth/sysadm/agould
- arn:aws:iam::123456789011:user/awsauth/drivera
- arn:aws:iam::123456789011:user/awsauth/jhsu

Groups:
- arn:aws:iam::123456789011:group/awsauth/all-users
- arn:aws:iam::123456789011:group/awsauth/orgadmins


_____________________
Account:    blee-dev
_____________________
Account:    blee-poc
_____________________
Account:    blee-prod
_____________________
Account:    master
Users:
- arn:aws:iam::222222222222:user/agould

Groups:
- arn:aws:iam::222222222222:group/Admins
```

Some variations:

```
$ awsauth report --users --full --account Managment
$ awsauth report --users --full
$ awsauth report --credentials --account Managment
$ awsauth report --credentials
```

6.1.2 Create an IAM user and group, and add the user to the group

Edit the following files:

- users.yaml
- groups.yaml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/groups.yaml b/groups.yaml
index 7f37144..d3fe879 100644
--- a/groups.yaml
+++ b/groups.yaml
@@ -46,3 +46,8 @@ groups:

+  - Name: testers
+    Ensure: present
+    Members:
+      - joeuser
+      - maryuser
diff --git a/users.yaml b/users.yaml
index 22d2d61..5424bf4 100644
--- a/users.yaml
+++ b/users.yaml
@@ -36,3 +36,6 @@ users:

+  - Name: joeuser
+    Email: joeuser@example.com
+    Team: test
+  - Name: maryuser
+    Email: maryuser@example.com
+    Team: test
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
```

6.1.3 Attach a IAM managed policy to your group

Edit file `groups.yaml`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/groups.yaml b/groups.yaml
index d3fe879..9e05738 100644
--- a/groups.yaml
+++ b/groups.yaml
@@ -50,4 +50,6 @@ groups:
   - Name: testers
     Ensure: present
     Members:
       - joeuser
       - maryuser
+    Policies:
+      - IAMReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
```

6.1.4 Attach a IAM custom policy to your group

Edit the following files:

- groups.yaml
- custom_policies.yaml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom_policies.yaml b/custom_policies.yaml
index da46ebb..5d411f0 100644
--- a/custom_policies.yaml
+++ b/custom_policies.yaml
@@ -111,3 +111,14 @@ custom_policies:
         Action:
           - aws-portal:Account*
         Resource: '*'
+
+  - PolicyName: ReadS3Bucket
+    Description: list and get objects from my s3 bucket
+    Statement:
+      - Effect: Allow
+        Action:
+          - s3:List*
+          - s3:Get*
+        Resource:
+          - arn:aws:s3:::my_bucket
+          - arn:aws:s3:::my_bucket/*
diff --git a/groups.yaml b/groups.yaml
index b506856..11e87cb 100644
--- a/groups.yaml
+++ b/groups.yaml
@@ -36,3 +36,4 @@ groups:
       - maryuser
     Policies:
       - IAMReadOnlyAccess
+      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
```

6.1.5 Modify attached custom policy

Edit file `custom_policies.yaml`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom_policies.yaml b/custom_policies.yaml
index d6f29d7..7f5748a 100644
--- a/custom_policies.yaml
+++ b/custom_policies.yaml
@@ -131,6 +131,8 @@ custom_policies:
          Resource:
            - arn:aws:s3:::my_bucket
            - arn:aws:s3:::my_bucket/*
+           - arn:aws:s3:::my_other_bucket
+           - arn:aws:s3:::my_other_bucket/*
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
$ aws iam get-policy-version --policy-arn <your_policy_arn> --version-id <version_id>
```

6.1.6 Detach policies, users from group

Edit the following files:

- groups.yaml

Example Diff:

```
(python3.6) ashely@horus:~/.awsorgs/spec.d> git diff
diff --git a/groups.yaml b/groups.yaml
index 9e05738..565b1ab 100644
--- a/groups.yaml
+++ b/groups.yaml
@@ -49,7 +49,4 @@ groups:
   - Name: testers
     Ensure: present
     Members:
-      - joeuser
-      - maryuser
     Policies:
-      - IAMReadOnlyAccess
-      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
```

6.1.7 Delete group, delete users

Files to edit:

- users.yaml
- groups.yaml

To delete IAM entities we must set attribute `Ensure: absent` to associated spec.

Example diff:

```
(python3.6) ashely@horus:~/.awsorgs/spec.d> git diff
diff --git a/groups.yaml b/groups.yaml
index 9e05738..4eda72b 100644
--- a/groups.yaml
+++ b/groups.yaml
@@ -47,9 +47,6 @@ groups:

   - Name: testers
-    Ensure: present
+    Ensure: absent
     Members:
     Policies:
diff --git a/users.yaml b/users.yaml
index 5424bf4..3e8b87d 100644
--- a/users.yaml
+++ b/users.yaml
@@ -37,5 +37,6 @@ users:
   - Name: joeuser
+    Ensure: absent
     Email: joeuser@example.com
     Team: test
   - Name: maryuser
+    Ensure: absent
     Email: maryuser@example.com
     Team: test
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
```

6.2 Cross Account Access Delegations - `awsauth delegations`

Prerequisites:

- IAM group with users to use as `TrustedGroup`

Commands used:

- git diff
- awsauth delegations
- awsauth delegations –exec
- awsauth report –roles

Spec files impacted:

- delegations.yaml
- custom_policies.yaml
- policy_sets.yaml

Actions:

- *Create a cross account access delegation*
- *Update the delegation to apply to all accounts*
- *Exclude some accounts from a delegation*
- *Attach a custom policy*
- *Modify a custom policy*
- *Create a policy set and apply it to the delegation*
- *Delete the delegation from all accounts*

6.2.1 Create a cross account access delegation

File to edit: delegations.yaml

- set `TrustedGroup` to your new group
- define a list of accounts in `TrustingAccount`
- define one managed policy in `Policies`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations.yaml b/delegations.yaml
index 1ae3245..4d571e9 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -101,3 +101,14 @@ delegations:

+  - RoleName: TestersRole
+    Ensure: present
+    Description: testing cross account delegation
+    TrustingAccount:
+    TrustedGroup: testers
+    RequireMFA: True
+    Policies:
+      - ReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
```

6.2.2 Update the delegation to apply to all accounts

File to edit: delegations.yaml

- set `TrustingAccount` to keyword `ALL`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations.yaml b/delegations.yaml
index 282db35..e46ac9e 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -104,14 +104,10 @@ delegations:
   - RoleName: TestersRole
     Ensure: present
     Description: testing cross account delegation
-    TrustingAccount:
-      - blee-dev
-      - blee-poc
-      - blee-prod
+    TrustingAccount: ALL
     TrustedGroup: testers
     RequireMFA: True
     Policies:
       - ReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
```

6.2.3 Exclude some accounts from a delegation

File to edit: delegations.yaml

- define a list of accounts in `ExcludeAccounts`

Example Diff:

```
:~/.awsorgs/spec.d> git diff
diff --git a/delegations.yaml b/delegations.yaml
index e46ac9e..8b01bb8 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -105,6 +105,10 @@ delegations:
     Ensure: present
     Description: testing cross account delegation
     TrustingAccount: ALL
+    ExcludeAccounts:
+      - blee-dev
+      - blee-prod
     TrustedGroup: testers
     RequireMFA: True
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
$ aws iam get-group-policy --group-name testers --policy-name DenyAssumeRole-
↪TestersRole
```

6.2.4 Attach a custom policy

Files to edit:

- custom_policies.yaml
- delegations.yaml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom_policies.yaml b/custom_policies.yaml
index 9399a60..a428164 100644
--- a/custom_policies.yaml
+++ b/custom_policies.yaml
@@ -120,3 +120,14 @@ custom_policies:
+
+  - PolicyName: ReadS3Bucket
+    Description: list and get objects from my s3 bucket
+    Statement:
+      - Effect: Allow
+        Action:
+          - s3:List*
+          - s3:Get*
+        Resource:
+          - arn:aws:s3:::my_bucket
+          - arn:aws:s3:::my_bucket/*
diff --git a/delegations.yaml b/delegations.yaml
```

(continues on next page)

(continued from previous page)

```
index 8b01bb8..ce9afa9 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -113,5 +113,6 @@ delegations:
     RequireMFA: True
     Policies:
       - ReadOnlyAccess
+      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|awsauth/ReadS3Bucket"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
$ aws iam get-group-policy --group-name testers --policy-name DenyAssumeRole-
↪TestersRole
```

6.2.5 Modify a custom policy

Files to edit:

- custom_policies.yaml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom_policies.yaml b/custom_policies.yaml
index a428164..7efe46b 100644
--- a/custom_policies.yaml
+++ b/custom_policies.yaml
@@ -131,3 +131,5 @@ custom_policies:
         Resource:
           - arn:aws:s3:::my_bucket
           - arn:aws:s3:::my_bucket/*
+          - arn:aws:s3:::my_other_bucket
+          - arn:aws:s3:::my_other_bucket/*
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles --full | grep -A12 awsauth/ReadS3Bucket
```

6.2.6 Create a policy set and apply it to the delegation

Files to edit:

- policy_sets.yaml
  - create a new policy_set:
    - use the same policies as are listed in the delegation
    - include a tag and value of your choice
- delegations.yaml
  - delete the `Policies` attribute from the delegation
  - set the `PolicySet` attribute to the name of your new policy set

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/policy_sets.yaml b/policy_sets.yaml
index ae4c72d..1d991d2 100644
--- a/policy_sets.yaml
+++ b/policy_sets.yaml
@@ -18,6 +18,14 @@ policy_sets:

+- Name: TesterPolicySet
+  Description: Access for testers
+  Tags:
+  - Key: jobfunctionrole
+    Value: True
+  Policies:
+  - ReadOnlyAccess
+  - ReadS3Bucket

diff --git a/delegations.yaml b/delegations.yaml
index 1ae3245..4d571e9 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -101,3 +101,14 @@ delegations:

   - RoleName: TestersRole
     Ensure: present
     Description: testing cross account delegation
     TrustingAccount:
     TrustedGroup: testers
     RequireMFA: True
-    Policies:
-      - ReadOnlyAccess
-      - ReadS3Bucket
+    PolicySet: TesterPolicySet
+      - ReadOnlyAccess
+      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ aws iam list-role-tags --role-name TestersRole
```

6.2.7 Delete the delegation from all accounts

Files to edit: delegations.yaml

- set `Ensure:  absent`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations.yaml b/delegations.yaml
index 2b050da..b6892d1 100644
--- a/delegations.yaml
+++ b/delegations.yaml
@@ -67,14 +67,10 @@ delegations:
        - ViewBilling

   - RoleName: TestersRole
-    Ensure: present
+    Ensure: absent
     Description: testing cross account delegation
     TrustingAccount: ALL
     ExcludeAccounts:
       - blee-poc
       - blee-dev
       - blee-prod
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|role/awsauth/ReadS3Bucket"
$ aws iam list-group-policies --group-name testers
```

6.3 IAM User Login Profiles - `awsloginprofile`

Prerequisites:

- admin access to auth account
- spec file setup
  - ~/.awsorgs/config.yaml for awsorgs configuration parameters
  - spec files in spec_dir directory which is defined in config.yaml
  - create at least one satelite account (see awsaccounts)

Commands used:

- awsauth users –exec
- awsloginprofile user –new
- awsloginprofile user –reset
- awsloginprofile user –report

spec files impacted:

- ~/.awsorgs/config.yaml
- spec_dir/users.yaml
- spec_dir/groups.yaml

Actions Summary:

- *Define org_access_role in ~/.awsorgs/config.yaml*
- *Assume OrgAdmin role for creating new IAM user*
- *Check current user login profile*
- *Create new IAM user in users.yaml and groups.yaml*
- *Create new IAM user with awsauth*
- *Create user login profile with awsloginprofile*
- *User will receive initial login instruction from email notification*
- *Check user login status*
- *Reset user login profile(password)*

6.3.1 Define org_access_role in `~/.awsorgs/config.yaml`

Edit ~/.awsorgs/config.yaml

```
org_access_role: awsauth/OrgAdmin
```

6.3.2 Assume `OrgAdmin` role for creating new IAM user

Assume auth acount administrtor role:

```
(py36) [jhsu@scrappy-aws awsorgs]$ aws-assume-role master-abc-OrgAdmin
(py36) [jhsu@scrappy-aws awsorgs]$
(py36) [jhsu@scrappy-aws awsorgs]$ aws-whoami
{
  "UserId": "AROAJS2JVTC6CC3YZX3BC:abc-admin@OrgAdmin",
  "Account": "123456789011",
  "Arn": "arn:aws:sts::123456789011:assumed-role/OrgAdmin/abc-admin@OrgAdmin"
}
```

6.3.3 Check current user login profile

Run 'awsloginprofile user'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-1
User:                   abcaws1-user-1
Arn:                    arn:aws:iam::123456789011:user/awsauth/abcaws1-user-1
User Id:                AIDAI5DX7YNIPTLGTQXZK
User created:           2019-01-03 01:28:59+00:00
Login profile created:  2019-01-10 19:32:11+00:00
Passwd reset required:  True
Password last used:     2019-01-10 19:55:35+00:00
Delegations:
```

(continues on next page)

(continued from previous page)

```
  Account Id      Alias                   Role
  2222222222      acct-abcaws1          awsauth/abcaws1


(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2
no such user: abcaws1-user-2
```

6.3.4 Create new IAM user in `users.yaml` and `groups.yaml`

Edit users.yaml

```
- Name: abcaws1-user-2
  Email: xyz@yahoo.com
  Team: team-abcaws1
```

Edit groups.yaml

```
- Name: group-abcaws1
  Members:
    - abcaws1-user-1
    - abcaws1-user-2
```

6.3.5 Create new IAM user with `awsauth`

Run 'awsauth users –exec'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsauth users  --exec
awsorgs.utils: INFO     Creating user 'abcaws1-user-2'
awsorgs.utils: INFO     arn:aws:iam::123456789011:user/awsauth/abcaws1-user-2
awsorgs.utils: INFO     Adding user 'abcaws1-user-2' to group 'all-users'
awsorgs.utils: INFO     Adding user 'abcaws1-user-2' to group 'group-abcaws1'
```

6.3.6 Create user login profile with `awsloginprofile`

Run 'wsloginprofile user –new'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2 --new
```

6.3.7 User will receive initial login instruction from email notification

Check email titled "login profile" for initial AWS login instruction

```
Dear User,

You have been granted access to our central AWS authentication account.  From here␣
↪you can assume designated roles into other AWS accounts in our Organization.

You must complete the following tasks to configure your access:

1) Use the credentials below to log into the AWS console.  You will be required to␣
↪change your password as you log in.  The rules for good passwords are as follows:
```

(continues on next page)

(continued from previous page)

```
- Minimum password length: 8
- Require at least one uppercase character from Latin alphabet. (A-Z)
- Require at least one lowercase character from Latin alphabet. (a-z)
- Require at least one symbol. (!@#$%^&amp;*()_+-=[]{}|')
- Require at least one number. (0-9)

IMPORTANT: your one time password will expire after 24 hours.

IAM User Name:        abcaws1-user-2
One Time Password:    Stroller_Ochre+402_Disputed
Login URL:            https://master-aaa.signin.aws.amazon.com/console
```

6.3.8 Check user login status

Run 'wsloginprofile user'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2

User:                   abcaws1-user-2
Arn:                    arn:aws:iam::123456789011:user/awsauth/abcaws1-user-2
User Id:                AIDAJKHIBNEWTQ3T2QOYC
User created:           2019-01-15 00:06:45+00:00
Login profile created:  2019-01-15 00:07:08+00:00
Passwd reset required:  False
Password last used:     2019-01-15 00:51:46+00:00
Delegations:
  Account Id      Alias                   Role
  222222222222    acct-abcaws1          awsauth/abcaws1
```

6.3.9 Reset user login profile(password)

Run 'wsloginprofile user –reset'

```
awsloginprofile abcaws1-user-2 --reset
```

CHAPTER 7

Functional tests for awsorgs/awsaccounts

Prerequisites:

- admin access to auth account
- spec file setup
  - ~/.awsorgs/config.yaml for awsorgs configuration parameters
  - spec files in spec_dir directory which is defined in config.yaml
  - create at least one satelite account (see awsaccounts)

7.1 AWS account alias

Commands used:

- awsaccounts alias
- awsaccounts alias –exec
- awsaccounts report

spec files impacted:

- ~/.awsorgs/config.yaml
- spec_dir/account-spec.yml

Actions Summary:

- Define org_access_role in ~/.awsorgs/config.yaml
- Define spec_dir/accoount-spec.yml
- awsaccoutns alias
- awsaccoutns alias –exec
- awsaccoutns report

7.1.1 Define org_access_role in ~/.awsorgs/config.yaml

Edit ~/.awsorgs/config.yaml

```
org_access_role: OrganizationAccountAccessRole
```

7.1.2 Show current awsaccoutns alias

Run 'awsacccouns report'

```
(py36) [jhsu@scrappy-aws ~]$ awsaccounts report


_________________________
Active Accounts in Org:

Name:                Alias               Id:             Email:
account-abcaws1   acct-abcaws1      123456789011    mail1@yahoo.com
account-abcaws2   acct-abcaws2      123456789011    mail2@yahoo.com
account-abcaws3   acct-abcaws3      123456789011    mail3@yahoo.com
```

7.1.3 Edit spec_dir/accoount-spec.yml

Change account-abcaws3 alias from acct-abcaws3 to accnt-abcaws3:

```
- Name: account-abcaws3
Team: team-abcaws3
Alias: accnt-abcaws3
Email: mail3@yahoo.com
```

7.1.4 Dryrun awsaccounts alias

Run 'awsaccount alias'

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts alias

[dryrun] awsorgs.utils: INFO     resetting account alias for account 'account-abcaws3
↪' to 'accnt-abcaws3'; previous alias was 'acct-abcaws3'
```

7.1.5 Exec awsaccounts alias

Run 'awsaccount alias –exec'

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts alias --exec

awsorgs.utils: INFO     resetting account alias for account 'account-abcaws3' to
↪'accnt-abcaws3'; previous alias was 'acct-abcaws3'
```

7.1.6 awsaccounts report

Run 'awsaccount report'

```
(py36) [jhsu@scrappy-aws doc]$ awsaccounts report


_________________________
Active Accounts in Org:

Name:               Alias               Id:             Email:
account-abcaws1   acct-abcaws1      123456789011    mail1@yahoo.com
account-abcaws2   acct-abcaws2      123456789011    mail2@yahoo.com
account-abcaws3   accnt-abcaws3     123456789011    mail3@yahoo.com
```

CHAPTER 8

Functional Tests for awsauth tool

Prerequisites:

- admin access to auth account
- spec file setup
  - install template spec files
  - spec files under git
  - site specific paramaters defined in common.yaml
  - configure ~.awsorgs/config.yaml
  - create at least one satelite account (see awsaccounts)

8.1 Users and Groups - `awsauth users`

Commands used:

- git diff
- awsauth users
- awsauth users –exec
- awsauth report –users

Spec files impacted:

- users-spec.yml
- groups-spec.yml
- custom-policy-spec.yml

Actions Summary:

- report IAM users and groups in accounts

- create an IAM user and group, and add the user to the group
- attach a IAM managed policy to your group
- attach a IAM custom policy to your group
- modify attached custom policy
- detach policies, users from group
- delete group, delete user

8.1.1 Report users and groups in all accounts

Run `awsauth report` command with `--users` flag:

```
$ awsauth report --users
_____________________________________________
IAM Users and Groups in all Org Accounts:
_____________________
Account:    Managment
Users:
- arn:aws:iam::123456789011:user/awsauth/sysadm/agould
- arn:aws:iam::123456789011:user/awsauth/drivera
- arn:aws:iam::123456789011:user/awsauth/jhsu

Groups:
- arn:aws:iam::123456789011:group/awsauth/all-users
- arn:aws:iam::123456789011:group/awsauth/orgadmins


_____________________
Account:    blee-dev
_____________________
Account:    blee-poc
_____________________
Account:    blee-prod
_____________________
Account:    master
Users:
- arn:aws:iam::222222222222:user/agould

Groups:
- arn:aws:iam::222222222222:group/Admins
```

Some variations:

```
$ awsauth report --users --full --account Managment
$ awsauth report --users --full
$ awsauth report --credentials --account Managment
$ awsauth report --credentials
```

8.1.2 Create an IAM user and group, and add the user to the group

Edit the following files:

- users-spec.yml
- groups-spec.yml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/groups-spec.yml b/groups-spec.yml
index 7f37144..d3fe879 100644
--- a/groups-spec.yml
+++ b/groups-spec.yml
@@ -46,3 +46,8 @@ groups:

+  - Name: testers
+    Ensure: present
+    Members:
+      - joeuser
+      - maryuser
diff --git a/users-spec.yml b/users-spec.yml
index 22d2d61..5424bf4 100644
--- a/users-spec.yml
+++ b/users-spec.yml
@@ -36,3 +36,6 @@ users:

+  - Name: joeuser
+    Email: joeuser@example.com
+    Team: test
+  - Name: maryuser
+    Email: maryuser@example.com
+    Team: test
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
```

8.1.3 Attach a IAM managed policy to your group

Edit file `groups-spec.yml`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/groups-spec.yml b/groups-spec.yml
index d3fe879..9e05738 100644
--- a/groups-spec.yml
+++ b/groups-spec.yml
@@ -50,4 +50,6 @@ groups:
   - Name: testers
     Ensure: present
     Members:
       - joeuser
       - maryuser
+    Policies:
+      - IAMReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
```

8.1.4 Attach a IAM custom policy to your group

Edit the following files:

- groups-spec.yml
- custom-policy-spec.yml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom-policy-spec.yml b/custom-policy-spec.yml
index da46ebb..5d411f0 100644
--- a/custom-policy-spec.yml
+++ b/custom-policy-spec.yml
@@ -111,3 +111,14 @@ custom_policies:
         Action:
           - aws-portal:Account*
         Resource: '*'
+
+  - PolicyName: ReadS3Bucket
+    Description: list and get objects from my s3 bucket
+    Statement:
+      - Effect: Allow
+        Action:
+          - s3:List*
+          - s3:Get*
+        Resource:
+          - arn:aws:s3:::my_bucket
+          - arn:aws:s3:::my_bucket/*
diff --git a/groups-spec.yml b/groups-spec.yml
index b506856..11e87cb 100644
--- a/groups-spec.yml
+++ b/groups-spec.yml
@@ -36,3 +36,4 @@ groups:
       - maryuser
     Policies:
       - IAMReadOnlyAccess
+      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
```

8.1.5 Modify attached custom policy

Edit file `custom-policy-spec.yml`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom-policy-spec.yml b/custom-policy-spec.yml
index d6f29d7..7f5748a 100644
--- a/custom-policy-spec.yml
+++ b/custom-policy-spec.yml
@@ -131,6 +131,8 @@ custom_policies:
          Resource:
            - arn:aws:s3:::my_bucket
            - arn:aws:s3:::my_bucket/*
+           - arn:aws:s3:::my_other_bucket
+           - arn:aws:s3:::my_other_bucket/*
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
$ aws iam get-policy-version --policy-arn <your_policy_arn> --version-id <version_id>
```

8.1.6 Detach policies, users from group

Edit the following files:

- groups-spec.yml

Example Diff:

```
(python3.6) ashely@horus:~/.awsorgs/spec.d> git diff
diff --git a/groups-spec.yml b/groups-spec.yml
index 9e05738..565b1ab 100644
--- a/groups-spec.yml
+++ b/groups-spec.yml
@@ -49,7 +49,4 @@ groups:
   - Name: testers
     Ensure: present
     Members:
-      - joeuser
-      - maryuser
     Policies:
-      - IAMReadOnlyAccess
-      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
$ aws iam list-attached-group-policies  --group-name testers
$ aws iam get-policy --policy-arn <your_policy_arn>
```

8.1.7 Delete group, delete users

Files to edit:

- users-spec.yml
- groups-spec.yml

To delete IAM entities we must set attribute `Ensure: absent` to associated spec.

Example diff:

```
(python3.6) ashely@horus:~/.awsorgs/spec.d> git diff
diff --git a/groups-spec.yml b/groups-spec.yml
index 9e05738..4eda72b 100644
--- a/groups-spec.yml
+++ b/groups-spec.yml
@@ -47,9 +47,6 @@ groups:

   - Name: testers
-    Ensure: present
+    Ensure: absent
     Members:
     Policies:
diff --git a/users-spec.yml b/users-spec.yml
index 5424bf4..3e8b87d 100644
--- a/users-spec.yml
+++ b/users-spec.yml
@@ -37,5 +37,6 @@ users:
   - Name: joeuser
+    Ensure: absent
     Email: joeuser@example.com
     Team: test
   - Name: maryuser
+    Ensure: absent
     Email: maryuser@example.com
     Team: test
```

Review proposed changes in `dry-run` mode:

```
$ awsauth users
```

Implement and review changes:

```
$ awsauth users --exec
$ awsauth report --users
```

8.2 Cross Account Access Delegations - `awsauth delegations`

Prerequisites:

- IAM group with users to use as `TrustedGroup`

Commands used:

- git diff
- awsauth delegations
- awsauth delegations –exec
- awsauth report –roles

Spec files impacted:

- delegations-spec.yml
- custom-policy-spec.yml

Actions:

- create a cross account access delegation
- update the delegation definition
- update attached custom policy
- delete delegation

8.2.1 Create a cross account access delegation

File to edit: delegations-spec.yml

- set `TrustedGroup` to your new group
- define a list of accounts in `TrustingAccount`
- define one managed policy in `Policies`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations-spec.yml b/delegations-spec.yml
index 1ae3245..4d571e9 100644
--- a/delegations-spec.yml
+++ b/delegations-spec.yml
@@ -101,3 +101,14 @@ delegations:

+  - RoleName: TestersRole
+    Ensure: present
+    Description: testing cross account delegation
+    TrustingAccount:
+    TrustedGroup: testers
+    RequireMFA: True
+    Policies:
+      - ReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
```

8.2.2 Update the delegation to apply to all accounts

File to edit: delegations-spec.yml

- set `TrustingAccount` to keyword `ALL`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations-spec.yml b/delegations-spec.yml
index 282db35..e46ac9e 100644
--- a/delegations-spec.yml
+++ b/delegations-spec.yml
@@ -104,14 +104,10 @@ delegations:
   - RoleName: TestersRole
     Ensure: present
     Description: testing cross account delegation
-    TrustingAccount:
-      - blee-dev
-      - blee-poc
-      - blee-prod
+    TrustingAccount: ALL
     TrustedGroup: testers
     RequireMFA: True
     Policies:
       - ReadOnlyAccess
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
```

8.2.3 Exclude some accounts from a delegation

File to edit: delegations-spec.yml

- define a list of accounts in `ExcludeAccounts`

Example Diff:

```
:~/.awsorgs/spec.d> git diff
diff --git a/delegations-spec.yml b/delegations-spec.yml
index e46ac9e..8b01bb8 100644
--- a/delegations-spec.yml
```

(continues on next page)

(continued from previous page)

```
+++ b/delegations-spec.yml
@@ -105,6 +105,10 @@ delegations:
     Ensure: present
     Description: testing cross account delegation
     TrustingAccount: ALL
+    ExcludeAccounts:
+      - blee-dev
+      - blee-prod
     TrustedGroup: testers
     RequireMFA: True
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|TestersRole"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
$ aws iam get-group-policy --group-name testers --policy-name DenyAssumeRole-
↪TestersRole
```

8.2.4 Attach a custom policy

Files to edit:

- custom-policy-spec.yml
- delegations-spec.yml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom-policy-spec.yml b/custom-policy-spec.yml
index 9399a60..a428164 100644
--- a/custom-policy-spec.yml
+++ b/custom-policy-spec.yml
@@ -120,3 +120,14 @@ custom_policies:
+
+  - PolicyName: ReadS3Bucket
+    Description: list and get objects from my s3 bucket
+    Statement:
+      - Effect: Allow
+        Action:
+          - s3:List*
+          - s3:Get*
+        Resource:
+          - arn:aws:s3:::my_bucket
+          - arn:aws:s3:::my_bucket/*
diff --git a/delegations-spec.yml b/delegations-spec.yml
index 8b01bb8..ce9afa9 100644
--- a/delegations-spec.yml
+++ b/delegations-spec.yml
```

(continues on next page)

(continued from previous page)

```
@@ -113,5 +113,6 @@ delegations:
     RequireMFA: True
     Policies:
       - ReadOnlyAccess
+      - ReadS3Bucket
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|awsauth/ReadS3Bucket"
$ aws iam list-group-policies --group-name testers
$ aws iam get-group-policy --group-name testers --policy-name AllowAssumeRole-
↪TestersRole
$ aws iam get-group-policy --group-name testers --policy-name DenyAssumeRole-
↪TestersRole
```

8.2.5 Modify a custom policy

Files to edit:

- custom-policy-spec.yml

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/custom-policy-spec.yml b/custom-policy-spec.yml
index a428164..7efe46b 100644
--- a/custom-policy-spec.yml
+++ b/custom-policy-spec.yml
@@ -131,3 +131,5 @@ custom_policies:
         Resource:
           - arn:aws:s3:::my_bucket
           - arn:aws:s3:::my_bucket/*
+          - arn:aws:s3:::my_other_bucket
+          - arn:aws:s3:::my_other_bucket/*
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles --full | grep -A12 awsauth/ReadS3Bucket
```

8.2.6 Delete the delegation from all accounts

Files to edit: delegations-spec.yml

- set `Ensure:  absent`

Example Diff:

```
~/.awsorgs/spec.d> git diff
diff --git a/delegations-spec.yml b/delegations-spec.yml
index 2b050da..b6892d1 100644
--- a/delegations-spec.yml
+++ b/delegations-spec.yml
@@ -67,14 +67,10 @@ delegations:
       - ViewBilling

   - RoleName: TestersRole
-    Ensure: present
+    Ensure: absent
     Description: testing cross account delegation
     TrustingAccount: ALL
     ExcludeAccounts:
       - blee-poc
       - blee-dev
       - blee-prod
```

Review proposed changes in `dry-run` mode:

```
$ awsauth delegations
```

Implement and review changes:

```
$ awsauth delegations --exec
$ awsauth report --roles  | egrep "^Account|role/awsauth/ReadS3Bucket"
$ aws iam list-group-policies --group-name testers
```

CHAPTER 9

Functional tests for awsorgs/awsloginprofile

Prerequisites:

- admin access to auth account
- spec file setup
  - ~/.awsorgs/config.yaml for awsorgs configuration parameters
  - spec files in spec_dir directory which is defined in config.yaml
  - create at least one satelite account (see awsaccounts)

9.1 AWS awsloginprofile

Commands used:

- awsauth users –exec
- awsloginprofile user –new
- awsloginprofile user –reset
- awsloginprofile user –report

spec files impacted:

- ~/.awsorgs/config.yaml
- spec_dir/users-spec.yml
- spec_dir/groups-spec.yml

Actions Summary:

- Define org_access_role in ~/.awsorgs/config.yaml
- Define spec_dir/users-spec.yml
- Define spec_dir/groups-spec.yml

- awsauth users –exec
- awsloginprofile user –new
- awsloginprofile user –reset
- awsloginprofile user –report

9.1.1 Define org_access_role in ~/.awsorgs/config.yaml

Edit ~/.awsorgs/config.yaml

```
org_access_role: awsauth/OrgAdmin
```

9.1.2 Assume OrgAdmin role for creating new IAM user

Assume auth acount administrtor role:

```
(py36) [jhsu@scrappy-aws awsorgs]$ aws-assume-role master-abc-OrgAdmin
(py36) [jhsu@scrappy-aws awsorgs]$
(py36) [jhsu@scrappy-aws awsorgs]$ aws-whoami
{
  "UserId": "AROAJS2JVTC6CC3YZX3BC:abc-admin@OrgAdmin",
  "Account": "123456789011",
  "Arn": "arn:aws:sts::123456789011:assumed-role/OrgAdmin/abc-admin@OrgAdmin"
}
```

9.1.3 Check current user login profile

Run 'awsloginprofile user'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-1
User:                   abcaws1-user-1
Arn:                    arn:aws:iam::123456789011:user/awsauth/abcaws1-user-1
User Id:                AIDAI5DX7YNIPTLGTQXZK
User created:           2019-01-03 01:28:59+00:00
Login profile created:  2019-01-10 19:32:11+00:00
Passwd reset required:  True
Password last used:     2019-01-10 19:55:35+00:00
Delegations:
  Account Id      Alias                   Role
  2222222222      acct-abcaws1          awsauth/abcaws1


(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2
no such user: abcaws1-user-2
```

9.1.4 Create new IAM user in user-spec.yml and group-spec.yml

Edit users-spec.yml

```
- Name: abcaws1-user-2
  Email: xyz@yahoo.com
  Team: team-abcaws1
```

Edit groups-spec.yml

```
- Name: group-abcaws1
  Members:
    - abcaws1-user-1
    - abcaws1-user-2
```

9.1.5 Create new IAM user with awsauth

Run 'awsauth users –exec'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsauth users  --exec
awsorgs.utils: INFO     Creating user 'abcaws1-user-2'
awsorgs.utils: INFO     arn:aws:iam::123456789011:user/awsauth/abcaws1-user-2
awsorgs.utils: INFO     Adding user 'abcaws1-user-2' to group 'all-users'
awsorgs.utils: INFO     Adding user 'abcaws1-user-2' to group 'group-abcaws1'
```

9.1.6 Create user login profile with awsloginprofile

Run 'wsloginprofile user –new'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2 --new
```

9.1.7 User will receive initial login instruction from email notification

Check email titled "login profile" for initial AWS login instruction

```
Dear User,

You have been granted access to our central AWS authentication account.  From here␣
↪you can assume designated roles into other AWS accounts in our Organization.

You must complete the following tasks to configure your access:

1) Use the credentials below to log into the AWS console.  You will be required to␣
↪change your password as you log in.  The rules for good passwords are as follows:

- Minimum password length: 8
- Require at least one uppercase character from Latin alphabet. (A-Z)
- Require at least one lowercase character from Latin alphabet. (a-z)
- Require at least one symbol. (!@#$%^&amp;*()_+-=[]{}|')
- Require at least one number. (0-9)

IMPORTANT: your one time password will expire after 24 hours.

IAM User Name:        abcaws1-user-2
One Time Password:    Stroller_Ochre+402_Disputed
Login URL:            https://master-aaa.signin.aws.amazon.com/console
```

9.1.8 Check user login status

Run 'wsloginprofile user'

```
(py36) [jhsu@scrappy-aws awsorgs]$ awsloginprofile abcaws1-user-2

User:                   abcaws1-user-2
Arn:                    arn:aws:iam::123456789011:user/awsauth/abcaws1-user-2
User Id:                AIDAJKHIBNEWTQ3T2QOYC
User created:           2019-01-15 00:06:45+00:00
Login profile created:  2019-01-15 00:07:08+00:00
Passwd reset required:  False
Password last used:     2019-01-15 00:51:46+00:00
Delegations:
  Account Id      Alias                   Role
  222222222222    acct-abcaws1          awsauth/abcaws1
```

9.1.9 Reset user login profile(password)

Run 'wsloginprofile user –reset'

```
awsloginprofile abcaws1-user-2 --reset
```

search